Simmani: Runtime Power Modeling for Arbitrary RTL with Automatic Signal Selection

Donggyu Kim*
University of California, Berkeley
dgkim@berkeley.edu

Jerry Zhao
University of California, Berkeley
jerryz123@berkeley.edu

Jonathan Bachrach
University of California, Berkeley
jrb@berkeley.edu

Krste Asanović
University of California, Berkeley
krste@berkeley.edu

ABSTRACT

This paper presents a novel runtime power modeling methodology which automatically identifies key signals for power dissipation of any RTL design. The toggle-pattern matrix is constructed with the VCD dumps from a training set, where each signal is represented as a high-dimensional point. By clustering signals showing similar switching activities, a small number of signals are automatically selected, and then the design-specific but workload-independent activity-based power model is constructed using regression against cycle-accurate power traces obtained from industry-standard CAD tools. We can also automatically instrument an FPGA-accelerated RTL simulation with runtime activity counters to obtain power traces of realistic workloads at speed. Our methodology is demonstrated with a heterogeneous processor composed of an in-order core and a custom vector accelerator, running not only microbenchmarks but also real-world machine-learning applications.

1 INTRODUCTION

As power and energy efficiency has been the primary concern for both low-power portable computers and high-end servers, runtime power estimation plays an important role not only in validation of hardware design ideas during the design process but also in effective runtime power, energy, and thermal optimizations and management. As a result, there has been significant prior work on various power-modeling methodologies.

Microarchitectural analytic power models [11, 28, 44, 45, 60, 66] are widely used in computer architecture research and early hardware design phases. These models in general rely on microarchitectural performance simulators [3, 7, 55, 67] to collect necessary statistics. This approach helps computer architects gain some high-level intuition before RTL implementation, but are limited to microarchitectures similar to those used to build the abstract model. Moreover, as simulation rate is a bottleneck with this methodology, computer architects are forced to sample and examine only a tiny fraction of real-world workloads for their studies. Finally, since power validation requires at least RTL implementation, constructing and validating power models is much more difficult for new types of application-specific accelerators.

Alternatively, power modeling using performance counters has been widely adopted for runtime power and thermal management in real microprocessors [4, 5, 8, 25, 46, 63]. Power models are constructed in terms of statistics from existing performance counters, and calibrated against power measurement from real systems. These power models provide quick power estimates by profiling full execution of applications, which can be used in runtime power and thermal optimizations such as dynamic voltage and frequency scaling (DVFS). However, this method has been only successful for well-known traditional microprocessors with existing real implementations. With a novel hardware design, designers should manually identify microarchitectural activities highly correlated with dynamic power dissipation, which is also extremely difficult for non-traditional hardware designs.

With the slow down in historical transistor scaling, the only way to sustain performance gain is through specialization with application-specific accelerators. Indeed, RTL implementation has become a standard procedure in computer architecture research to estimate the area, power, and energy for novel design ideas. However, dynamic power dissipation is not one-dimensional and cannot be statically determined as it depends heavily on signal activities that can vary across different workloads. Moreover, runtime power, energy, and thermal-management techniques should be studied for novel hardware designs to improve their energy efficiency. For this reason, a general, accurate, and efficient runtime power modeling methodology is required for future architecture research.

In this paper, we present Simmani, a novel activity-based runtime power modeling methodology that automatically selects the key signals for power dissipation in any RTL design. Our methodology was inspired by the observation that *signals showing similar toggle patterns have similar effect on dynamic power dissipation.* In the power modeling flow, the toggle pattern matrix, where each RTL signal is represented as a high-dimensional point, is constructed from VCD dumps generated from RTL simulation of the training set. As similarities of signals are quantified by the Euclidean distances between two points, a small number of signals are selected through clustering with dimensionality reduction. Then, the power model is trained through regression against cycle-accurate power traces from industry-standard CAD tools.

*Now at Apple

MICRO-52, October 12–16, 2019, Columbus, OH, USA

ACM ISBN 978-1-4503-6938-1/19/10…$15.00
https://doi.org/10.1145/3352460.3358322

We also present a technique to automatically instrument the target RTL design with toggle activity counters for FPGA-accelerated RTL simulation, which enables runtime power analysis for non-trivial workloads. By applying compiler passes using the FIRRTL compiler [27], Simmani automatically inserts activity counters for the selected signals that can be sampled by the software simulation driver for runtime power estimation.

The main contributions of this paper are as follows:

- Simmani is a general and easy-to-use runtime power modeling methodology for any RTL design. From VCD dumps and power traces, Simmani automatically constructs power models by selecting a small number of signals without requiring designers' intuition.
- Simmani trains and validates power models against RTL designs with cycle-accurate power traces from industry-standard CAD tools, which gives a high confidence for the model. Simmani also uses standard statistical methods to minimize modeling errors.
- Simmani enables fast runtime power estimation in FPGA-accelerated simulations by automatically instrumenting any RTL design with activity counters, dramatically reducing designers' manual effort. Fast power simulation also enables various case studies including power/thermal analysis of custom accelerators for emerging applications.

2 RELATED WORK

2.1 Microarchitectural Power Modeling

Analytical power modeling [11, 28, 44, 45, 60, 66] combined with microarchitectural software simulators [3, 7, 55, 67] is widely-used for computer architecture research. This method enables early architecture-level design-space exploration, helping designers gain high-level intuitions before RTL implementation. However, the power models need to be strictly validated against RTL implementations or real systems, which is difficult when exploring new non-traditional designs. We believe Simmani will help improve pre-RTL power models for novel hardware designs by discovering necessary modeling variables for future accelerator research.

2.2 Power Model Validation

Shafi et al. [59] validate an event-driven power model against the IBM PowerPC 405GP processor. Mesa-Martinez et al. [49] validate power and thermal models by measuring the temperature of real machines. The authors measure temperature using an infrared camera and translate temperature to power using a genetic algorithm. Xi et al. [69] validate McPAT against the IBM POWER7 processor and illustrate how inaccuracies can arise without careful tuning and validation. Lee et al. [41] propose a regression-based calibration of McPAT against existing processors to improve its prediction accuracy. McKeown et al. [47] characterize power and energy of an open-source 25-core processor from its silicon implementation. However, these methodologies can only be applied using existing machines or proprietary data. Jacobson et al. [28] suggest a power model from pre-defined microarchitectural events and validate it against RTL simulation. However, the approach relies on designer annotations and microbenchmarks exploiting familiarity with a particular family of processor architectures.

2.3 Runtime Power Modeling with Performance Counters

Power modeling based on performance-monitoring counters is also popular for power estimation [4, 5, 8, 25, 46, 63]. This method provides a quick power estimate, which is also useful for runtime power/thermal optimizations, by profiling full execution of applications. In addition, LeBeane et al. [37] show a power modeling technique that maps platform-specific event counters to McPAT's event counts.

There are also a variety of studies on phase/kernel-based power modeling. Isci et al. [26] characterize power phases with event counter statistics collected by dynamic binary instrumentation. Zheng et al. [70] present a cross-platform phase-based power modeling methodology that predicts the target design's power from the host platform's counter statistics. Wu et al. [68] and Greathouse et al. [20] develop a GPGPU performance and power modeling methodology that clusters training kernels based on performance scaling behaviors and classifies the group of a new kernel with neural nets based on performance counter values.

However, these methodologies are limited to well-known microprocessors with existing silicon implementations. For novel hardware designs, computer architects need intuition to define representative microarchitectural events highly correlated with power dissipation, which is extremely difficult without collecting empirical data from multiple costly tape-outs.

Unlike the previous work on event-based power modeling, Simmani trains high-fidelity runtime power models by *automatically selecting a small number of signals for any RTL designs using industry-standard CAD tools.* In addition, activity counters collecting signal statistics are automatically inserted in FPGA-accelerated simulations to provide rapid power estimates. We also believe our work can bootstrap various counter/phase-based power modeling efforts, by hinting at what activity counters should be available in novel hardware designs for emerging applications.

2.4 Statistical Performance/Power Modeling

There is significant previous work on statistical performance/power modeling for uniprocessors [16, 29, 30, 38, 39] and chip multiprocessors [23, 32, 40]. Regression [16, 29, 38–40] or neural network [23, 30, 32] models based on microprocessor microarchitectural parameters are trained from simulations of a small number of configurations to predict performance and power for unseen configurations without detailed simulations.

However, all these models are constructed in the microprocessor context. For non-traditional hardware designs, high-level microarchitectural parameters must be carefully identified using designers' intuition.

2.5 Cycle-Level RTL Power Modeling

Activity-based cycle-level RTL power modeling was also explored in previous work. Metha et al. [48] build table-based power models for small-size modules by clustering their input transitions resulting in similar energy dissipation to reduce the number of entries in the table. Our approach is different in that we cluster signals based on their toggle patterns to choose a small number of signals as regression variables. Gupta et al. [21] construct four-dimensional

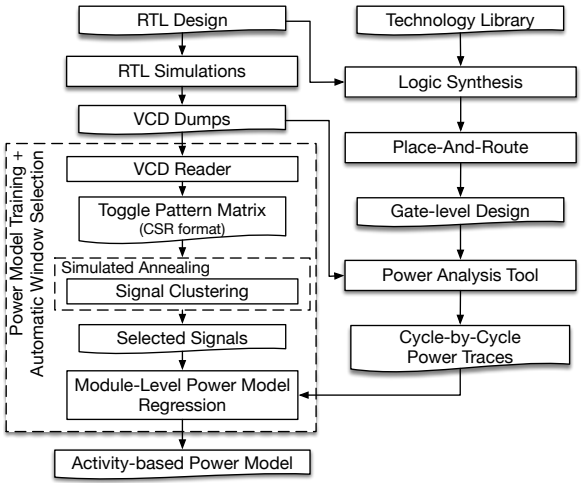


Figure 1: Power modeling flow

table-based macro models for combinational logic indexed by input/output signal switching activities.

Bogliolo et al. [10] build regression-based RTL power models in terms of input and output signals of combinational macro blocks divided by registers. This approach is not scalable since all switching activities of registers need to be tracked, which is intractable for complex hardware designs. In contrast, Simmani is scalable as it automatically selects a small number of signals from a large-scale design for power-model regression.

2.6 FPGA-Based Power/Energy Modeling

There has also been extensive work on accelerating power and energy modeling using FPGAs. Coburn et al. [13] instrument the target design with power computing units for each RTL library component, which incurs significant FPGA resource overhead for large hardware designs. Ghodrat et al. [19] improve this methodology to reduce the FPGA resource overhead by splitting the instrumented target design across the FPGA and software, which may significantly slow down the emulation speed without careful partitioning.

Bhattacharjee et al. [6] manually instrument event counters to collect statistics from the FPGA for runtime power modeling. Sunwoo et al. [64] manually instrument cycle-level power models that are trained with manually selected signals. Even though these methodologies are applied to fairly large hardware designs, they do not generalize for any RTL design, require designer intuition and manual effort as 1) microarchitectural events and RTL signals must be manually identified and 2) the target design must be manually instrumented with event counters. In contrast, Simmani automatically identifies the RTL signals most correlated with power dissipation and automatically adds activity counters to collect statistics from FPGA-accelerated simulations.

Zoni et al. [71] select signals from input/output signals in the module boundaries of the design hierarchy, construct a linear power model in terms of these signals, and instrument the runtime power model visible by software. In general, input and output signals are not the most correlated with power dissipation of a given module, and thus, a smaller number of internal signals are preferred to a larger number of input and output signals for accurate power modeling.

Kim et al. [35] propose sample-based energy modeling using FPGAs. In their methodology, any RTL design is instrumented with scan chains and I/O trace buffers to snapshot the RTL state from FPGAs. By randomly sampling RTL state snapshots from realistic applications running on complex hardware designs and replaying the snapshots on gate-level simulation, the average power and the energy of the whole execution can be accurately computed with the confidence interval. However, this methodology does not provide runtime power estimation, which is crucial for power phase analysis and runtime power optimization techniques. Moreover, the instrumentation overhead is much larger than just adding event counters for a small number of signals. We instead use this framework to provide ground truth for power-model training and validation with non-trivial workloads.

3 POWER MODEL TRAINING

In this paper, we present the Simmani framework, which automatically selects signals most correlated with power dissipation and trains power models in terms of the selected signals for any RTL design. The core idea is to *cluster signals showing similar toggle patterns to choose distinctive signals*, and then, *train power models in terms of these selected signals using cycle-accurate power traces*. The intuition is that signals showing similar toggle patterns have similar effect on dynamic power dissipation and can be *factored* to share the same coefficient in the power model, minimizing modeling error. Figure 1 describes the overall power modeling flow in the Simmani framework.

Section 3.1 introduces the power modeling background. Section 3.2 explains how the toggle patten matrix is constructed from VCD dumps. Section 3.3 describes how important signals for power dissipation are found through clustering. Section 3.4 explains how the number of signals is determined through model selection with simulated annealing. Section 3.5 explains how to obtain detailed cycle-accurate power traces from commercial CAD tools. Section 3.6 shows how module-level power models using the selected signals are trained through regression against cycle-accurate power traces. Section 3.7 presents how the window size for the toggle pattern matrix is automatically decided.

3.1 Power Modeling Background

CMOS power consumption can be decomposed into three major factors:

$$P_{total} = P_{dyn} + P_{dp} + P_{leak} = \alpha f(C_L V_{DD}^2 + V_{DD} I_{peak} t_s) + V_{DD} I_{leak}$$

The dynamic power, P_{dyn}, is consumed when the capacitance, C_L, is charged or discharged, while the direct-path power, P_{dp}, is consumed during rise/fall times due to short-circuit current, $I_{peak} t_s$, when transistors are switching. Both cause power dissipation when signals *toggle*, the ratio of which is captured by the activity factor, α. The leakage power, P_{leak}, is, on the other hand, consumed due to leakage current, I_{leak}, even when transistors are not switching.

We may assume leakage power is constant under the condition that the temperature is well-controlled and the threshold voltage does not change dynamically. In this case, the leakage power can

be statically computed by CAD tools. In addition, the direct-path power is minimized by CAD tools, and thus, much smaller than dynamic power. However, dynamic power, a primary factor in power dissipation, is hard to determine statically since the activity factor is highly workload-dependent. For this reason, static block-level dynamic power estimation from CAD tools can easily be pessimistic or optimistic. Therefore, we should collect activity statistics from simulations to estimate the dynamic power dissipation of each workload.

Dynamic power can be computed by summing signal toggle densities over all CMOS gates [53]:

$$P_{dyn} = \frac{1}{2} V_{DD}^2 \sum_{g \in \{gates\}} C_g D_g$$

where C_g and D_g are the load capacitance and the toggle density of gate g, respectively. Unfortunately, such toggle densities are only available through extremely detailed gate-level simulation, which is not practical for collecting related statistics from real-world workloads running on complex hardware designs.

Therefore, for large-scale designs, we approximate the dynamic power in terms of event statistics associated with their effective capacitances:

$$P_{dyn} \approx \frac{1}{2} V_{DD}^2 \sum_{e \in \{events\}} C_e D_e$$

where C_e and D_e are the effective capacitance and the statistics of event e, respectively.

Microarchitectural power models such as Wattch [11] and McPAT [45] analytically compute capacitances for regular structures [54] and collect manually identified event statistics from microarchitectural software simulators [7, 55, 67] before RTL implementation. Performance-counter-based power modeling [4, 5, 8, 25, 46, 63] uses existing counters in the system, and finds the effective capacitance of each counter event through regression against power measurement of the real machine. These methodologies have been effective for well-known traditional microarchitectures.

However, for arbitrary novel designs, these approaches are very challenging as 1) manually selecting important signal/event activities is difficult and 2) finding the effective capacitance is also difficult. In the following sections, we tackle both problems for any RTL design automatically and systematically.

3.2 Toggle-Pattern Matrix from VCD Dumps

The first step for power-model training is to construct the *toggle-pattern matrix* using VCD dumps from RTL simulations of the training set. For accurate power modeling, we carefully choose small workloads that represent real-world applications. If the training set is too small, the trained model cannot accurately predict power consumption of unseen workloads. If the training set is too large, the model training is bottlenecked by RTL simulation and power analysis tools that need to process a large volume of VCD dumps. In this paper, we choose ISA tests and microbenchmarks and replays of random sample snapshots from long-running applications, as an initial attempt, because these workloads highly utilize processors with a variety of operations. Synthesizing more representative workloads remains as the future work as discussed in Section 6.

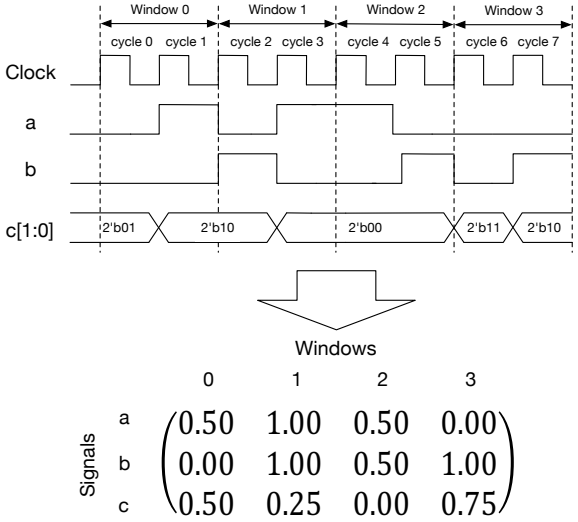


Figure 2: Constructing a toggle-pattern matrix.

The toggle-pattern matrix is a collection of toggle-density vectors of all signals in the RTL design. Each element of this matrix is constructed as follows:

$$v_{ij} = \frac{\text{total number of toggles of signal } i \text{ over window } j}{\text{width of signal } i \times \text{window size}}$$

where v_{ij} is the element at row i and column j of the toggle pattern matrix.

Figure 2 shows a simple example of how the toggle-pattern matrix is constructed from VCD dumps with a window size of two cycles. For single-bit signals, the number of toggles is just the number of value transitions. For example, the total number of value transitions of signal a over window 0 is 2, and thus, $v_{a0} = 2/2 = 1.0$. The other elements for signal a and b are computed in the same way.

For multi-bit busses, the number of toggles is the Hamming distance between the value at the previous cycle and the value at the current cycle. For example, the Hamming distance of signal c at cycle 1 is 2. The reason each matrix element is divided by the width of the signal is we want to group busses of different widths together in the same cluster if they show similar toggle patterns. Hence, $v_{c0} = 2/(2 \times 2) = 0.5$.

The toggle-pattern matrix is very large for a complex hardware design. But most entries in this matrix are zeros for a typical hardware design, since only a small number of signals tend to be active in a given time slot. Therefore, the toggle-pattern matrix is represented as a sparse matrix using the compressed sparse row (CSR) format.

The similarity of two signals is measured by the Euclidean distance between two vectors. It is intuitive that two signals having a short distance between them have a similar effect on power dissipation. In this case, the window size plays an important role in quantifying similarities. Determining the window size is discussed in Section 3.7.

3.3 Automatic Signal Selection

Once the toggle-pattern matrix is constructed, we want to partition signals into a handful of groups, each of which collects signals showing similar toggle patterns. Since the similarity is measured as the Euclidean distance between two signal vectors, this problem is identical to a *clustering problem*.

There are several challenges in signal clustering. First, exact clustering is known as an NP-hard problem, and thus, a randomized algorithm such as `k-means` should be used, where clustering results are different with different initial seeds. Moreover, with a non-trivial hardware design, there are a large number of signals, each of which is represented as a very high-dimensional point, which makes clustering more challenging. Specifically, if we have signal traces of N cycles with window size w, the dimension of each signal is N/w, which can be easily a very large number with a long trace.

Spectral clustering is a class of algorithms for clustering of high-dimensional data points through dimensionality reduction [9]. In this paper, we reduce the dimension of data by projecting data into principal components from singular vector decomposition (SVD). The algorithm to partition the data set into k clusters is as follows:

(1) Find the space V spanned by the top k right singular vectors from SVD.
(2) Project the data points into V through matrix multiplication.
(3) Cluster the projected points through `k-means++` [1], which selects better initial seeds than random initial centroids.
(4) Repeat multiple times and select the clustering with the best score.

It is also proven that the projection brings the data points closer to their cluster centers [1]. In addition, this algorithm can be efficiently implemented with high-performance linear algebra libraries.

Once signal clustering is done, *the signals that are the closest to the center of each cluster are selected*, which will be regression variables in power model training. The rationale is these signals have the smallest variance of similarities to other points in the same cluster, and thus, we expect them to introduce the smallest errors in regression than any other signals.

3.4 Finding the Number of Signals

The clustering algorithm in Section 3.3 finds the optimal clustering when the number of signals is given, but does not determine the number of clusters. In many cases, it is hard to know in advance how many signals should be selected for power modeling with an arbitrary hardware design. We want to select as many signals as possible for accurate power modeling, but not too many signals to avoid model overfitting and to enable power model instrumentation (Section 4).

Finding the number of signals is the same as a *model selection problem*. We want to select the best clustering among candidate models for a given data. The idea is we run the clustering algorithm with different numbers of clusters and find the one having the best objective score.

For model selection, we use the Bayesian Information Criterion (BIC) [58], which is commonly used beyond the hypothesis tests. The BIC is a penalized model-fit statistic as it prefers a model having less parameters to a model having more parameters but only fitting marginally better. [2] Formally, for model M_j, the BIC is formulated as follows:

$$BIC_j = p_j ln(n) - 2ln(L_j)$$

where n is the number of points in the data, and p_j and L_j are the size and the likelihood of model M_j, respectively. The absolute value of the BIC is barely interpretable. However, the difference of values, $\Delta BIC = BIC_{new} - BIC_{old}$, is of interest. For example, we determine the new model is very strong compared to the old model if $\Delta BIC < -10$ [3].

For clustering, we use the formula derived by Pelleg and Moore [57], assuming underlying distributions are spherical Gaussians. The maximum likelihood estimate for the variance is:

$$\hat{\sigma}^2 = \frac{1}{n-k} \sum_{i=1}^{n} \|x_i - \mu(x_i)\|^2$$

where k is the number of clusters and $\mu(x_i)$ is the cluster center of x_i. Intuitively, this quantity explains how far points in each group are scattered away from their cluster center.

Then, the log likelihood of model M_j is the summation of log likelihoods of all clusters:

$$ln(L_j) = -\frac{n}{2} ln(2\pi) - \frac{nd}{2} ln(\hat{\sigma}^2) - \frac{n-k}{2} + \sum_{i=1}^{k} n_i ln(\frac{n_i}{n})$$

where d is the dimension of points and n_i is the number of points in cluster i. The number of parameters, p_j, is $(k-1) + dk + 1$ for $(k-1)$ cluster probabilities, k centroids of dimension d, and one variance estimate. Intuitively, this quantity expresses how well signals in each group are clustered around their cluster center. This metric is also used by SimPoint [61] to find the optimal clustering for program phases.

To select the number of signals, we keep track of ΔBIC by increasing the number of clusters, k. To avoid getting stuck at local minima, we employ *simulated annealing* as follows:

(1) Run the clustering algorithm with the initial k, and compute the BIC, which is the initial best clustering.
(2) Increase k, and run the clustering algorithm and compute the BIC.
(3) If $\Delta BIC = BIC_{cur} - BIC_{best} < -10$, update the best clustering, and go to 2.
(4) Otherwise, decrease the temperature, T, and go to 2 with the probability of $exp(\frac{\Delta BIC}{T})$.

This algorithm starts with a high temperature, which gradually decreases over iterations. Therefore, this algorithm is not likely to terminate in early iterations even if no better clustering is found, helping escape from local minima. However, with low temperatures, the algorithm has a very high probability to terminate in later iterations as the current best clustering is very likely to be the global optimum.

[1] For the theorem and its proof, refer to [9]

[2] Compared to the Akaike information criterion (AIC), the BIC assigns more penalties on the number of parameters, having higher chances to reject models with more signals.

[3] The Bayes factor is equal to $\exp(-\Delta BIC/2)$

3.5 Obtaining Cycle-Accurate Power Traces

For accurate power modeling for RTL designs, cycle-accurate power traces are necessary. We obtain these power traces using commercial CAD tools as shown in Figure 1. We first obtain the gate-level design from the target RTL using logic synthesis[4] with the target technology library[5]. Clock gating is also automatically inferred during synthesis. Since commercial SRAM compilers were not available, we characterize SRAMs used in the target design with CACTI 6.5 [52], and generate library files using commercial library compilers[6]. To obtain accurate estimates for the timing, area, and the floorplan of the final silicon, we also place and route the post-synthesis design[7].

After place-and-route, the commercial power analysis tool[8] can compute cycle-accurate power traces from RTL VCD dumps. In this detailed power analysis, RTL signal activities are propagated into gate-level signals and the cycle-by-cycle power for modules in the design is computed. Since the full cycle-by-cycle power traces are generated instead of just the average power, this process tends to be the bottleneck for power modeling.

Throughout this paper, we assume the cycle-accurate power traces obtained in this section are the true power for training and evaluation.

3.6 Power-Model Regression

Once n signals are automatically selected (Section 3.3), we train the power model in terms of these signals against the cycle-accurate power traces from commercial CAD tools (Section 3.5). As discussed in Section 3.1, power-model regression finds the effective capacitances for the signal activities. We are also interested in module-level power modeling for thermal analysis [62].

Formally, for each module k, we want to find a function f_k that accurately approximates the actual power dissipation in terms of signal activities:

$$p_{kj} \approx f_k(x_{1j}, x_{2j}, \cdots, x_{nj})$$

for all time window j, where p_{kj} and x_{ij} are the power consumption of module k and the toggle density of signal i in window j, respectively. The total power consumption of the target design in window j is just the sum of power consumptions in window j over all modules.

Power models need to be as simple as possible to minimize the computation overhead for runtime power and thermal analysis. On the other hand, power models need to be more complex than linear regression since, in general, power dissipation is not a linear function of the activities of the selected signals. To cope with non-linearity, we use linear regression with interactions and high-order terms. One justification is that, theoretically, a non-linear function can be approximated with its Taylor expansion with polynomial terms. There is also a large amount of empirical evidence that linear regression with polynomial terms of manually selected events and signals is reasonably accurate for microprocessors [5, 8, 25, 28, 40, 64]. Lastly, these interactions and high-order terms can be viewed as an approximation to *hidden activities* that are not solely captured by the selected signals.

Therefore, we also assume power dissipation is a function of the following form:

$$\begin{aligned} p_{kj} = \alpha + \beta_1 x_{1j} + \beta_2 x_{2j} + \cdots \beta_n x_{nj} + \\ \beta_{11} x_{1j}^2 + \beta_{22} x_{2j}^2 + \cdots + \\ \beta_{12} x_{1j} x_{2j} + \cdots + \beta_{123} x_{1j} x_{2j} x_{3j} + \cdots \end{aligned}$$

where α and β's are parameters to be trained. As there are an infinite number of terms in the Taylor expansion, we limit the order of terms to two. However, the number of terms still grows exponentially with the number of signals. For instance, if 50 signals are selected, there will be 2550 terms in the model. Linear regression with this many terms tends to be unstable and suffers from high variance, losing prediction accuracy.

Moreover, models with a large number of variables are less interpretable, as well as increasing the compute overhead. From the perspective of activity-based power modeling, each regression variable represents a certain activity in the design and its coefficient is its effective capacitance. However, all these activities are not equally important for power modeling across different modules. Indeed, we want to systematically select most of the single-order terms but only a small number of higher-order terms to correlate between signal activities and power consumption without prior knowledge of these signals.

The previous two issues can be viewed as a problem of *regularization* and *variable selection* in linear regression. Prediction accuracy can be improved by shrinking coefficients through regularization with penalized regression, which reduces the variance of coefficients while trading off the bias. Variable selection can further improve the prediction accuracy, preventing overfitting, as well as the interpretability.

In this paper, we employ the elastic net [72] for both regularization and variable selection. The elastic net is penalized regression with a convex combination of the L1 and L2 penalties of coefficients. As a result, the elastic net behaves mostly like LASSO [65], while preserving the prediction power of ridge regression.

We assume the training data is standardized having the zero mean and the unit standard deviation before regression. Then, to find the coefficients β with given power trace $\mathbf{p}$ and toggle densities $\mathbf{X}$, the elastic net solves the following optimization problem:

$$\min_{\beta} \left\{ \frac{1}{2n} ||\mathbf{p} - \mathbf{X}\beta||^2 + \lambda \left(\frac{1-\rho}{2} ||\beta||^2 + \rho ||\beta||_1 \right) \right\} \quad (1)$$

where ρ and λ are determined by K-fold cross-validation, a technique that splits the training data into K groups for both training and validation. We also restrict that all coefficients are non-negative[9]. This optimization can be efficiently solved by coordinate descent [18]. Notice that ridge regression and LASSO are special instances of the elastic net when $\rho = 1$ and $\rho = 0$, respectively.

When we apply the elastic net for power modeling, many unimportant variables are desirably eliminated as shown in Section 5.2.

[4] We used Synopsys Design Compiler version O-2018.06-SP4
[5] We used the TSMC 45nm technology
[6] We used Synopsys Library Compiler version J-2014.09-SP4 and Synopsys Milkyway version J-2014.09-SP4
[7] We used Synopsys IC Compiler version O-2018.06-SP4
[8] We use Synopsys PrimeTimePX version O-2018.06-SP4

[9] We do not have this constraint on uncore in our example target design, whose power is given by subtracting the sum of power of all other modules from the total power.

3.7 Finding the Window Size

As alluded in Section 3.2, the window size plays an important role in quantifying similarities in the toggle-pattern matrix. If the window size is too small, two very similar signals (e.g. the input and the output of shift registers) may have a large distance between them. On the other hand, if the window size is too large, two distinctive signals may appear similar. Therefore, the window size affects the number of selected signals and in turn prediction accuracies.

The window size is dependent on the target design *and* the training data set. We also observed that the clustering algorithm tends to select more signals with a larger window size. This is because a shorter window size dramatically increases distances between points, and thus, having more clusters does not help improving the quality of clustering.

Indeed, manual selection of the window size is another challenge and requires many trials and errors. Instead, we propose automatic signal selection as follows:

(1) For a given window size,
  (a) Select signals with clustering (Section 3.3 and 3.4).
  (b) Train a power model for each submodule (Section 3.6) and compute its BIC.
(2) Select the window size that minimizes the total score of power models.

Note that Step 1 can be parallelized for different window sizes to minimize runtime overhead as trainings are independent of one another. For the score of each power model in Step 2, we use the BIC for linear regression:

$$BIC = \frac{\sum_i^N error_i^2}{\sigma^2} + ln(N) \cdot df$$

where $error_i$ is the error of each data point i, df is the degree of the freedom of the model, which is a function of λ in Equation (1), and N and σ^2 are the size and the variance of data, respectively. Intuitively, the BIC finds the model with small errors as well as a small number of variables.

Computing exact df for the elastic net is computationally expensive. However, when λ in Equation (1) is small, which is the case when only a small number of variables are selected, df is very close to the degree of the freedom of LASSO, which is equal to the number of nonzero coefficients [73]. Therefore, we approximate df with the number of nonzero coefficients in the model when we compute the BIC.

Since we have multiple power models for each submodule in the design, we may want to select the new window over the old window if the geometric mean of all Bayes factors [31] of each model is greater than 1, which is expressed as follows:

$$\sqrt[K]{\prod_{k=1}^{K} \exp\left(\frac{-\Delta BIC^k}{2}\right)} = \exp\left(\frac{-\sum_{k=1}^{K} \Delta BIC^k}{2K}\right) > 1$$

$$\Leftrightarrow \sum_{k=1}^{K} \Delta BIC^k = \sum_{k=1}^{K} BIC_{new}^k - \sum_{k=1}^{K} BIC_{old}^k < 0$$

where K is the number of models and BIC^k is the BIC of model k. Therefore, we select the window size that minimizes the sum of all BICs of each model.

Section 5.3 presents evaluations on how window sizes affect the number of selected signals and the accuracy of power models.

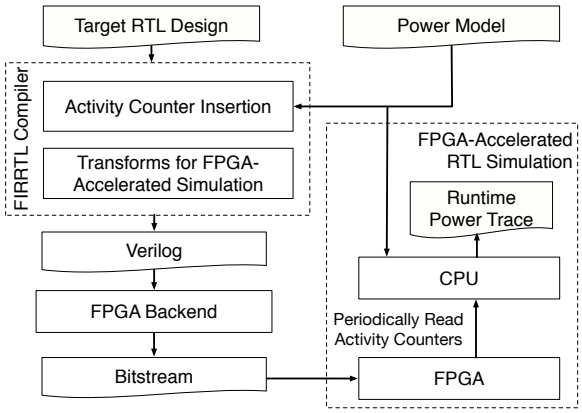


Figure 3: Tool flow for runtime power analysis with FPGAs

4 POWER MODEL INSTRUMENTATION

Once the power model is trained as in Section 3, the target RTL needs to be instrumented for runtime power analysis and evaluation. The target RTL design is automatically instrumented with the power model using custom transforms inserted in the FIRRTL compiler as shown in Figure 3. Section 4.1 overviews the FIRRTL compiler that enables hardware designers to write custom compiler passes for any RTL design. Section 4.2 shows how activity counters collecting the toggle activities of the selected signals are automatically inserted into the target RTL by a custom compiler pass. Section 4.3 shows how runtime power traces are obtained from FPGA-accelerated RTL simulation for various case studies.

4.1 The FIRRTL Compiler

Software compilers like LLVM have their own intermediate representations (IRs) to let software engineers write custom transforms and instrumentation. Likewise, the FIRRTL compiler [27] provides an IR for hardware designs. By writing custom transforms that operate on any hardware design represented by this IR, hardware designers can avoid design-specific engineering effort. We wrote custom compiler passes applicable to any hardware design to instrument activity counters and transform the target design for runtime power analysis on the FPGA.

For now, the FIRRTL compiler supports Chisel designs only. However, note that this framework is language-agnostic. Once there is a front-end to translate a HDL to FIRRTL, this framework can be reused for any hardware design in the HDL, significantly improving productivity [10].

4.2 Activity Counter Insertion

As shown in Figure 3, activity counters are automatically inserted by the compiler pass using the information from the power model. Figure 4 shows components instrumented by the compiler pass to collect the toggle activities of the selected signals.

For each selected signal, the HD unit is inserted to compute the Hamming distance between the value at the current cycle and the value at the previous cycle. For a single-bit signal, it is just an XOR gate. For a multi-bit bus, the HD unit computes XORs of individual

[10] The Verilog frontend is in progress

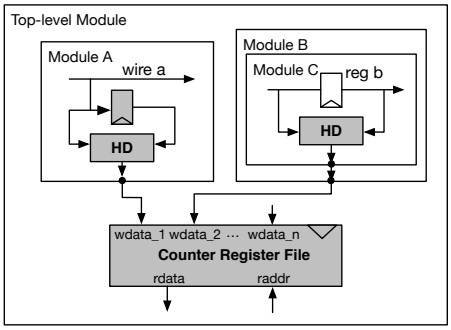


Figure 4: Activity counter instrumentation. Gray boxes are automatically instrumented by the compiler pass.

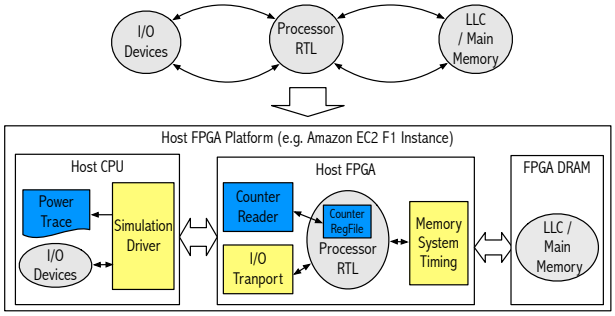


Figure 5: Mapping the target system to the host platform.

bits and counts the number of 1's. If the selected signal is a wire, a shadow register that keeps the value at the previous cycle is also inserted, and then the input and the output of this shadow register are fed into the HD unit. On the other hand, if the selected signal is a register, we do not need a shadow register. Instead, the input and the output of the selected register are connected to the HD unit.

We also need counters that increment by the Hamming distance on each cycle. For this purpose, a counter register file is instantiated in the top-level module, with the number of write ports equal to the number of selected signals. The HD units in submodules are connected to the counter register file across different module hierarchies, and thus, the compiler pass creates module ports along the connections to the write ports.

The counter register file has one read port and can be visible as architectural state for software running in the target design. Alternatively, this read port can be directly connected to the top-level I/O for FPGA-accelerated RTL simulation as explained in Section 4.3.

4.3 Runtime Power Analysis with FPGAs

FPGA-accelerated RTL simulation is the only viable way for performance, power, and energy evaluation of complex RTL designs running real-world software before tape-out. The Strober framework [34, 35] automatically generates FPGA-accelerated RTL simulators from any RTL designs with custom compiler passes. We use this framework to obtain runtime power traces from FPGAs. Once the activity counters are inserted, the instrumented target design is consumed by the following custom transforms for FPGA-accelerated RTL simulation (Figure 3).

After the FPGA-accelerated RTL simulator is compiled into the bitstream, it is run on the FPGA along with the software simulation driver. Figure 5 shows how the transformed target design is mapped to the host platform. The processor RTL is mapped into the FPGA while the data for the last-level cache (LLC) and the DRAM are mapped into the FPGA DRAM. For the timing of the memory system, we have an abstract timing model that only keeps the tags of the LLC on the FPGA. The software driver with abstract I/O devices runs on the host CPU. The processor RTL infrequently communicates with the I/O devices through the I/O transport unit on the FPGA only when necessary (e.g. console I/O), minimizing simulation slowdown.

To obtain power traces, the simulation driver periodically reads the activity counter values through the activity counter unit on the FPGA, which is connected to the read port of the counter register file. When the counter values are read, the simulation is stalled so that it does not change the behavior of the target system. Counter sampling is infrequent, and thus, the simulation driver infrequently polls the counter read unit that only pauses the simulation when the counters are sampled.

After activity statistics are collected from the FPGA, the software driver, which has the power model information, performs the rest of computations for model-level power and dumps runtime power values to a file. As such, we obtain the power traces over the whole execution of real-world applications at the end of the simulation.

We also estimate the power dissipation of the LLC and the DRAM with event counters in the memory timing model. For the LLC, we characterize its read and write energy per access as well as its static power with CACTI [52], and collect the number of read/write accesses to the LLC. For the DRAM, we assume Micron's LPDDR2 SDRAM S4 [51] and use the spreadsheet power calculator provided by Micron [50] with statistics on read/write operations and row activations of the DRAM.

5 EVALUATION

5.1 Experimental Setup

Most of the power training algorithm (Section 3) is implemented in Python with the `SciPy`'s sparse matrix libraries, while the toggle matrix construction algorithm (Section 3.2) is implemented in C++. We import `k-means++` (Section 3.3) and the elastic net solver (Section 3.6) from `scikit-learn` [56].

Parameter	Rocket+Hwacha
Processor	Rocket 5-stage in-order processor
Accelerator	2048-bit wide vector unit and 64-bit wide scalar unit
Registers	32(int)/32(fp)/64(scalar)/256(vector)
L1 I and D $	Capacity: 32 KiB, Associativity: 8 ways
ITLB & DTLB	Reach: 128 KiB, Associativity: fully-associative
L2 TLB	Reach: 4 MiB, Associativity: direct-mapped
Cycle time	1 ns
Area	1.79 mm x 1.52 mm
L2 $	Capacity: 1 MiB, Latency: 23 cycles, Read energy: 116.1 pJ, Write energy: 95.9 pJ, Leakage power: 21.0 mW
DRAM	Latency: 80 cycles, Number of banks: 8, Number of rows in each bank: 16K, Open-page policy

Table 1: Parameters for Rocket+Hwacha

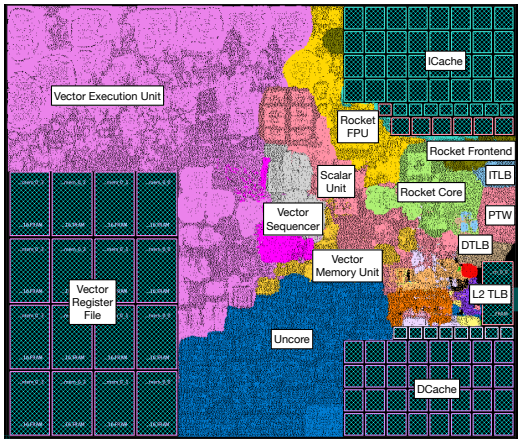


Figure 6: Floorplan of Rocket+Hwacha

The Simmani framework is demonstrated with the heterogeneous processor composed of the Rocket in-order core[11] [2] and the Hwacha vector accelerator[12] [42, 43] (Rocket+Hwacha). Table 1 shows its configuration for the evaluation. To best of our knowledge, no activity-based power model is developed for this design. We have abstract timing models for the L2 cache and the DRAM since we do not have corresponding RTL implementations for now. Even though we present only Rocket+Hwacha in this paper, we have also evaluated the BOOM out-of-order processor [12] as well as Rocket running the SPEC 2006/2017 integer benchmark suites [33].

The cycle time, area, and the floorplan of each processor are obtained from Synopsys Design Compiler (logic synthesis) and Synopsys IC Compiler (place-and-route) with the TSMC 45nm technology. Figure 6 shows the floorplan result of Rocket+Hwacha.

For power model training, we ran CAD tools for a day and the training algorithm for a half day on a high-performance server [13].

For FPGA-based simulation, we use AWS F1 instances. The FPGA-based simulator was synthesized at the frequency of 75 MHz, which took 6 hours, but the simulation rate for the case study was 39.3 MHz on average due to the overhead of counter sampling. For accurate validation, we carefully matched the timing of the memory system and the I/O devices between FPGA-based RTL simulation and software RTL simulation.

The training data set consists of 1) ISA tests, 2) microbenchmarks with their small input sets, and 3) 200 random sample snapshots from each benchmark of SqueezeNet with two images (dog and mousetrap).

For power model validation in Section 5.4, we used microbenchmarks with their large input sets. For case study in Section 5.5, we used a different set of 11 images for each benchmark of SqueezeNet.

5.2 Signal and Variable Selection

Table 2 shows the results of automatic signal selection by the algorithm in Section 3.3 with the average power and the standard deviation of each module for the training set. We counted all signals and data busses shown in the VCD dump except intermediate signals generated by the FIRRTL compiler (starting with `_GEN_`). Our signal clustering algorithm selected 113 signals out of 115,285 signals.

Total number of RTL signals	115,285				
Module	**Selected Signals**	**Single-order Terms**	**Second-order Terms**	**Average Power (mW)**	**Standard Dev (mW)**
Total	113	72	599	143.22	14.40
Rocket Fetch Unit	18	14	123	3.84	1.19
Rocket Core	27	30	134	7.55	2.24
Rocket FPU	1	7	72	2.00	1.87
Scalar Unit	1	16	70	3.43	1.16
Vector Sequencer	17	13	16	4.69	2.80
Vector Register File	1	24	74	36.30	4.57
Vector FP MulAdd	7	14	41	1.76	3.74
Vector Execute Unit	10	27	114	24.48	3.25
Vector Memory Unit	-	13	22	2.24	0.48
L1 ICache & ITLB	8	15	79	16.38	6.31
L1 DCache & DTLB	19	12	72	9.76	4.69
Uncore	4	15	75	30.77	3.58

Table 2: Results of automatic signal and variable selection

At first glance, it was surprising that only one signal was selected for the vector register file even though it dissipates a significant amount of power. If we had selected signals manually, the enable signals for this module would have been our primary choice. However, it turns out that those signals were clustered together with related signals in other modules but are not selected as representative signals. For example, the vector regfile write-enable and mask signals were clustered with a control signal of the floating point multiply-add unit, which was a representative signal of that cluster. Similarly, signals in the vector memory unit were clustered with signals in the load-store units of the vector execution unit, while signals in the FPU were clustered with signals in the Rocket core.

Table 2 also presents the variable selection results from power-model regression (Section 3.6). Note that some of terms appear across multiple modules, and thus, the total number of terms is smaller than the summation of the number of terms from each submodules. Our power modeling keeps 671 terms in total out of 6,554 candidate terms for training.

We also notice that cross-order terms can capture events across different modules. For example, our power modeling finds the interaction between a predicate signal in the vector execute unit and a hit signal in the data cache, which has a positive effect on the power dissipation of the vector register file.

5.3 Automatic Window Size Selection

Figure 7 shows how the window size affects the number of selected signals and the geometric mean of the R^2 values, a statistic for how well the model fits the training data, across module-level power models. First of all, more signals are selected as the window size increases. This is because a bigger window size makes data points closer, and therefore, having more clusters improves the quality of clustering. We can also see that this effect diminishes as the window size gets bigger.

Another trend is selecting more signals does not necessarily result in more accurate models. A model fits the training set well if its R^2 value is closer to 1.0[14]. As seen in Figure 7, the geometric mean of R^2 is the max at the window size of 340 cycles, and the sum of BICs, the score for window size selection in Section 3.7, is

[11] Commit: 50bb13d7887e5f9ca192431234b057ae9d8edb6c
[12] Commit: 519ed1642674909d89769eae1bd4fc35fa383e49
[13] Intel Xeon 32-core CPU @ 3.2 GHz with 25 MB L3 cache and 256 GB main memory.

[14] However, we do not use R^2 for model selection because a high R^2 may result from overfitting

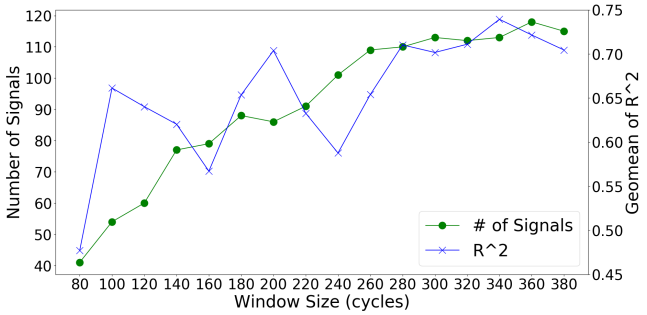


Figure 7: The number of selected signals and the geometric mean of R^2 across module-level power models for different window sizes

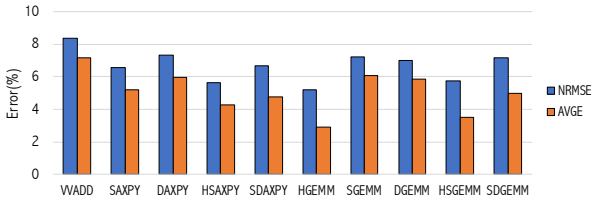


Figure 8: Power prediction errors for microbenchmarks

also the smallest at this point. Therefore, our training algorithm selects 113 signals with the window size of 340 cycles.

5.4 Power Model Validation

In this validation, we used well-known floating-point microbenchmarks vectorized for Rocket+Hwacha with various precisions. We estimated runtime power by sampling activity counters from the FPGA every 128 cycles, and compared it against power traces from Synopsys PrimeTime PX.

We computed the normalized mean-squared errors (NMSREs) and the average errors (AVGEs) across benchmarks. For N samples, NRMSEs and AVGEs are calculated as follows:

$$NRMSE = \frac{\sqrt{\sum_i^N (p_i - p_i^{pred})^2 / N}}{p_{avg}}, \quad AVGE = \frac{|p_{avg} - p_{avg}^{pred}|}{p_{avg}}$$

The NRMSE accounts for point-by-point errors while the AVGE cares about the average values only. The NRMSE also tends to be bigger than the AVGE.

Figure 8 shows both errors are within 9 % and our power modeling is reasonably accurate for these microbenchmarks.

5.5 Case Study

In this section, we demonstrate how Simmani can be used for custom accelerators targeting emerging applications in the HW/SW co-design flow. As an example of embedded vision applications, we use SqueezeNet [22], a neural network for image classification that achieves AlexNet-level accuracy with very small models. We evaluated three versions of SqueezeNet. In addition to the base variant (`SqueezeNet`), we evaluated a variant with 8-bit weight quantization (`SqueezeNet-8bits`), and a variant with both quantization and compressed weight storage (`SqueezeNet-Comp`).

We ported and vectorized SqueezeNet so that these three benchmarks can run on Rocket+Hwacha. For FPGA-based simulation, we

Benchmark	Cycles per inference (B)		Speedup
	Scalar	Vector	
SqueezeNet	22.89	1.58	14.45
SqueezeNet-8bits	26.53	1.57	16.94
SqueezeNet-Comp	16.02	1.37	11.72

Table 3: Performance of Rocket+Hwacha for SqueezeNet

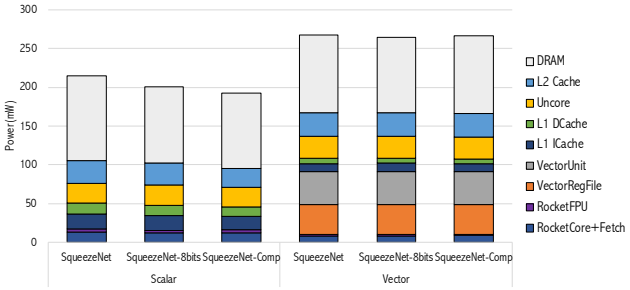


Figure 9: Power breakdown for SqueezeNet

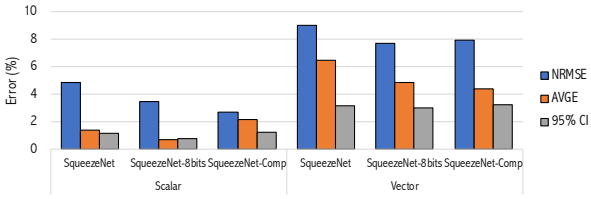
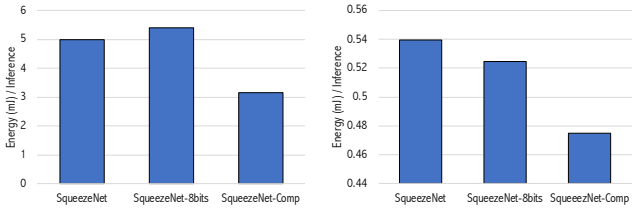


Figure 10: Power prediction errors for SqueezeNet



(a) Scalar **(b) Vector**

Figure 11: Energy efficiency for SqueezeNet

made initramfs Linux images that contain SqueezeNet binaries as well as 11 images for inference. To obtain power traces, we sampled counters from the FPGA every 100K cycles. For comparison, we also evaluated unoptimized scalar SqueezeNet benchmarks, which do not utilize the vector unit at all.

Table 3 shows the performance of Rocket+Hwacha for these three benchmarks without and with vectorization. First of all, vectorization significantly improves its performance. Without vectorization, quantization decreases the performance, while with vectorization, it marginally improves the performance. However, in both cases, there is a significant performance improvement with compression on top of vectorization.

Figure 9 shows the average power breakdowns for SqueezeNet. For the scalar benchmarks, we assume the vector accelerator is perfectly power-gated. Without vectorization, both quantization and compression reduce power consumption as they require less memory accesses. Surprisingly, Simmani reveals that Rocket+Hwacha consumes almost the same power across the vectorized benchmarks. This is because the vector accelerator is highly utilized during inferences thanks to small model sizes.

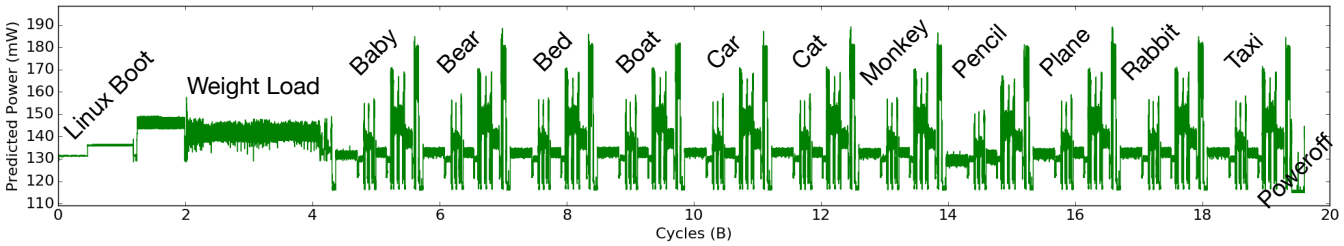


Figure 12: Power trace of Rocket+Hwacha for `SqueezeNet-Comp`

Figure 10 shows the power prediction errors with the 95 % confidence intervals. To validate the power estimates in Figure 9, we took random 50 sample snapshots of 1024 cycles from each benchmark. When each of these random snapshots was taken, its runtime power was estimated by sampling activity counters for this period of 1024 cycles. After the FPGA-based simulation was done, the power estimate of each sample point was obtained from sample replays, which was in turn compared against its runtime power estimate from the FPGA to compute the NRSE for 50 sample points. For the AVGE, we compared the average over the whole power trace against the average power estimate from sample replays, which also provided its confidence interval. From this validation, we can see that our power modeling also achieves good prediction accuracy for SqeeuzeNet.

Figure 11 shows the energy efficiency for SqueezNet. First of all, we can significantly improve energy efficiency with vectorization, which achieves significant speedups despite the increase in power consumption. Also, with vectorization, both quantization and compression gain energy efficiency as they require the same-level of power consumption.

Figure 12 shows the entire power trace without L2 and DRAM power for the vectorized `SqueezeNet-Comp` benchmark, while booting Linux, loading the model weights, and running inferences for 11 images over 20 billion cycle, which took 14 minutes on the FPGA but can take weeks or months with the CAD tools. We can also detect power phase changes over the whole execution of the benchmark, which will improve the effectiveness of runtime power/energy management techniques. We can also observe that power is more variable in the later layers than in the earlier layers for each inference because the later layers are more memory-intensive.

6 CONCLUSION AND FUTURE WORK

In this paper, we presented Simmani, a novel runtime power modeling methodology for any RTL designs by automatically selecting key signals for power dissipation. We also automatically instrumented the target design with activity counters to collect statistics from FPGAs, without requiring manual effort. We demonstrated Simmani's power modeling capabilities for non-traditional RTL designs with a case study of HW/SW co-design for machine-learning applications. Simmani is open-source [15] so that our methodology can be easily integrated into various accelerator research projects.

This section further discusses Simmani's potential use cases and enhancements.

Thermal Analysis. As we can obtain module-by-module power traces from FPGA-based simulation as well as floorplans from CAD tools (Figure 6), we can conduct pre-silicon thermal analysis for novel hardware designs running real-world workloads. HotSpot [62] is one example framework for thermal analysis. We plan to integrate HotSpot into Simmani for runtime thermal analysis with FPGA-based simulation.

Dynamic Power/Thermal Optimization. Runtime techniques for power and thermal management have been widely studied in the context of CPUs (e.g. [14, 15, 17, 24, 62, 63]). As custom accelerators are prevalent in computer systems, it is also important to do research on these techniques in the context of a variety of accelerators. As Simmani is generic for any hardware designs, we believe Simmani will be a useful tool for activity-based runtime power/thermal techniques for custom accelerators.

Power Model Composition. In this paper, Simmani is demonstrated for a relatively smaller hardware design with a single tile compared to contemporary heterogeneous multi-core SoCs. In heterogeneous multi-core systems, cores(tiles) and uncore are fairly independent blocks, and thus, we will improve Simmani's scalability with *power model composition*: we will train individual power models for each core(tile) and uncore, and then compose the total power with statistical methods. Lee et al. [40] also present such a methodology.

Automatic Training Set Generation. For Simmani, it is crucial to have a good training set for both signal selection and power model regression. In many cases, it is even more challenging to find a good training set that is fully representative for their real-world applications.

A good training set should have *good coverage of valid signal activities*. In fact, this challenge is also shared with input generation for simulation-based verification. Our future work will tackle this problem in a general setting for both hardware verification and power modeling. We believe workload generation with *coverage-based fuzzing* such as [36] is one promising approach.

ACKNOWLEDGMENT

The information, data, or work presented herein was funded in part by the Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award Number DE-AR0000849. Research was partially funded by ADEPT Lab industrial sponsors and affiliates Intel, Apple, Futurewei, Google, and Seagate, and by RISE Lab sponsor Amazon Web Services. Donggyu Kim was supported in part by the Kwanjeong Educational Foundation. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

[15] https://simmani.github.io

REFERENCES

[1] David Arthur and Sergei Vassilvitskii. 2007. K-Means++: the Advantages of Careful Seeding. In *ACM-SIAM Symposium on Discrete Algorithms*. https://doi.org/10.1145/1283383.1283494 arXiv:1212.1121

[2] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. 2015. *The Rocket Chip Generator.* Technical Report UCB/EECS-2016-17.

[3] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt. 2009. Analyzing CUDA workloads using a detailed GPU simulator. In *ISPASS*.

[4] Frank Bellosa. 2000. The benefits of event: driven energy accounting in power-sensitive systems. In *The 9th ACM SIGOPS European Workshop*.

[5] R. Bertran, M. Gonzelez, X. Martorell, N. Navarro, and E. Ayguade. 2013. A Systematic Methodology to Generate Decomposable and Responsive Power Models for CMPs. *IEEE Trans. Comput.* 62, 7 (Jul 2013), 1289–1302.

[6] Abhishek Bhattacharjee, Gilberto Contreras, and Margaret Martonosi. 2008. Full-system chip multiprocessor power evaluations using FPGA-based emulation. In *ISLPED*.

[7] Nathan Binkert, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, David A. Wood, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, and Tushar Krishna. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39 (Aug 2011).

[8] W. Lloyd Bircher and Lizy K. John. 2007. Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events. In *ISPASS*.

[9] Avrim Blum, John Hopcroft, and Ravindran Kannan. 2016. Foundations of data science. *Vorabversion eines Lehrbuchs* (2016).

[10] Alessandro Bogliolo, Luca Benini, and Giovanni De Micheli. 2000. Regression-based RTL power modeling. *ACM Transactions on Design Automation of Electronic Systems* 5 (2000).

[11] David Brooks, Vivek Tiwari, and Margaret Martonosi. 2000. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA*.

[12] Christopher Celio, Pi-Feng Chiu, Borivoje Nikolic, David A. Patterson, and Krste Asanović. 2017. BOOMv2: an open-source out-of-order RISC-V core. In *First Workshop on Computer Architecture Research with RISC-V (CARRV)*.

[13] Joel Coburn, Srivaths Ravi, and Anand Raghunathan. 2005. Power emulation: a new paradigm for power estimation. In *DAC*.

[14] Ryan Cochran, Can Hankendi, Ayse Coskun, and Sherief Reda. 2011. Pack & Cap: Adaptive DVFS and thread packing under power caps. In *MICRO*. ACM, 175–185.

[15] Qingyuan Deng, David Meisner, Abhishek Bhattacharjee, Thomas F. Wenisch, and Ricardo Bianchini. 2012. CoScale: Coordinating CPU and memory system DVFS in server systems. *MICRO* (2012).

[16] Christophe Dubach, Timothy M. Jones, and Michael F.P. O'Boyle. 2007. Microarchitectural design space exploration using an architecture-centric approach. In *MICRO*.

[17] Stijn Eyerman and Lieven Eeckhout. 2010. A counter architecture for online DVFS profitability estimation. *IEEE Trans. Comput.* 59, 11 (2010), 1576–1583.

[18] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. 2010. Regularization Paths for Generalized Linear Models via Coordinate Descent. *Journal of Statistical Software* 33, 1 (2010), 1–22.

[19] Mohammad Ali M.A. Ghodrat, Kanishka Lahiri, and Anand Raghunathan. 2007. Accelerating system-on-chip power analysis using hybrid power estimation. In *DAC*.

[20] Joseph L. Greathouse and Gabriel H. Loh. 2018. Machine learning for performance and power modeling of heterogeneous systems. In *ICCAD*.

[21] Subodh Gupta and Farid N. Najm. 2000. Power modeling for high-level power estimation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 8, 1 (2000), 18–29.

[22] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size.* Technical Report. http://arxiv.org/abs/1602.07360

[23] Engin Ïpek, Sally A. McKee, Rich Caruana, Bronis R. de Supinski, and Martin Schulz. 2006. Efficiently exploring architectural design spaces via predictive modeling. In *ASPLOS*.

[24] Canturk Isci, Alper Buyuktosunoglu, Chen Yong Cher, Pradip Bose, and Margaret Martonosi. 2006. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *MICRO*. 347–358.

[25] C. Isci and M. Martonosi. 2003. Runtime power monitoring in high-end processors: Methodology and empirical data. In *MICRO*.

[26] Canturk Isci and Margaret Martonosi. 2006. Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques. In *HPCA*. 122–133.

[27] Adam Izraelevitz, Jack Koenig, Patrick S. Li, Richard Lin, Angie Wang, Albert Magyar, Donggyu Kim, Colin Schmidt, Chick Markley, Jim Lawson, and Jonathan Bachrach. 2017. Hardware Reusability is FIRRTL Ground: Hardware Construction Languages, Compiler Frameworks, and Transformations. In *ICCAD*.

[28] Hans Jacobson, Alper Buyuktosunoglu, Pradip Bose, Emrah Acar, and Richard Eickemeyer. 2011. Abstraction and microarchitecture scaling in early-stage power modeling. In *HPCA*.

[29] P.J. Joseph, Kapil Vaswani, and Matthew J. Thazhuthaveetil. 2006. Construction and Use of Linear Regression Models for Processor Performance Analysis. In *HPCA*.

[30] P. J. Joseph, Kapil Vaswani, and Matthew J. Thazhuthaveetil. 2006. A predictive performance model for superscalar processors. In *MICRO*.

[31] E Kass, R and E Raftery, A. 1995. Bayes factors. *J. Amer. Statist. Assoc.* 90, 1995 (1995), 773–795.

[32] Salman Khan, Polychronis Xekalakis, John Cavazos, and Marcelo Cintra. 2007. Using Predictive Modeling for Cross-Program Design Space Exploration in Multicore Systems. In *PACT*.

[33] Donggyu Kim. 2019. *FPGA-Accelerated Evaluation and Verification of RTL Designs.* Ph.D. Dissertation. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-57.html

[34] Donggyu Kim, Christopher Celio, David Biancolin, Jonathan Bachrach, and Krste Asanović. 2017. Evaluation of RISC-V RTL with FPGA-Accelerated Simulation. In *First Workshop on Computer Architecture Research with RISC-V (CARRV)*.

[35] Donggyu Kim, Adam Izraelevitz, Christopher Celio, Hokeun Kim, Brian Zimmer, Yunsup Lee, Jonathan Bachrach, and Krste Asanović. 2016. Strober : Fast and Accurate Sample-Based Energy Simulation for Arbitrary RTL. In *ISCA*.

[36] Kevin Laeufer, Jack Koenig, Donggyu Kim, Jonathan Bachrach, and Koushik Sen. 2018. RFUZZ: coverage-directed fuzz testing of RTL on FPGAs. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.

[37] Michael LeBeane, Jee Ho Ryoo, Reena Panda, and Lizy Kurian John. 2015. WattWatcher: Fine-Grained Power Estimation for Emerging Workloads. In *International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*.

[38] Benjamin C Lee and David M Brooks. 2006. Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction. In *ASPLOS*.

[39] Benjamin C Lee and David M Brooks. 2007. Illustrative Design Space Studies with Microarchitectural Regression Models. In *HPCA*.

[40] Benjamin C. Lee, Jamison Collins, Hong Wang, and David Brooks. 2008. CPR: Composable performance regression for scalable multiprocessor models. In *MICRO*.

[41] Wooseok Lee, Youngchun Kim, Jee Ho Ryoo, Dam Sunwoo, Andreas Gerstlauer, and Lizy K John. 2015. PowerTrain: A learning-based calibration of McPAT power models. In *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*.

[42] Yunsup Lee, Albert Ou, Colin Schmidt, Sagar Karandikar, Howard Mao, and Krste Asanović. 2015. *The Hwacha Microarchitecture Manual, Version 3.8.1.* Technical Report UCB/EECS-2015-263. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-263.html

[43] Yunsup Lee, Colin Schmidt, Albert Ou, Andrew Waterman, and Krste Asanović. 2015. *The Hwacha Vector-Fetch Architecture Manual, Version 3.8.1.* Technical Report UCB/EECS-2015-262. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-262.html

[44] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi. 2013. GPUWattch: enabling energy optimizations in GPGPUs. In *ISCA*.

[45] Sheng Li, Jung Ho Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, and N.P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*.

[46] Tao Li and Lizy Kurian John. 2003. Run-time modeling and estimation of operating system power consumption. In *SIGMETRICS*.

[47] Michael McKeown, Alexey Lavrov, Mohammad Shahrad, Paul J. Jackson, Yaosheng Fu, Jonathan Balkind, Tri M. Nguyen, Katie Lim, Yanqi Zhou, and David Wentzlaff. 2018. Power and Energy Characterization of an Open Source 25-Core Manycore Processor. In *HPCA*.

[48] Huzefa Mehta, Robert Michael Owens, and Mary Jane Irwin. 1996. Energy characterization based on clustering. In *DAC*.

[49] Francisco Javier Mesa-Martinez, Joseph Nayfach-Battilana, and Jose Renau. 2007. Power model validation through thermal measurements. *ISCA*.

[50] Micron Technology. [n. d.]. Mobile LPDDR2 System-Power Calculator. https://www.micron.com/support/tools-and-utilities/power-calc. https://www.micron.com/support/tools-and-utilities/power-calc

[51] Micron Technology. 2012. *Micron Mobile LPDDR2 SDRAM S4.* Datasheet. Micron Technology.

[52] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. 2009. *CACTI 6.0 : A Tool to Model Large Caches.* Technical Report HPL-2009-85.

[53] F.N. Najm. 1994. A survey of power estimation techniques in VLSI circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 2, 4 (dec 1994), 446–455.

[54] Subbarao Palacharla, Norman P. Jouppi, and J. E. Smith. 1997. Complexity-effective superscalar processors. In *ISCA*.

[55] Avadh Patel, Furat Afram, and Shunfei Chen. 2011. MARSSx86: A full system simulator for x86 CPUs. In *DAC*.

[56] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[57] D. Pelleg and A.W. Moore. 2000. X-means: Extending K-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*.

[58] Gideon Schwarz. 1978. Estimating the Dimension of a Model. *The Annals of Statistics* 6, 2 (mar 1978), 461–464.

[59] H Shafi, P J Bohrer, J Phelan, C A Rusu, and J L Peterson. 2003. Design and validation of a performance and power simulator for PowerPC systems. *IBM Journal of Research and Development* 47, 5.6 (2003), 641–651.

[60] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2014. Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *ISCA*.

[61] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. 2002. Automatically characterizing large scale program behavior. In *ASPLOS*.

[62] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. 2003. Temperature-aware microarchitecture. In *ISCA*.

[63] Bo Su, Junli Gu, Li Shen, Wei Huang, Joseph L. Greathouse, and Zhiying Wang. 2014. PPEP: Online Performance, Power, and Energy Prediction Framework and DVFS Space Exploration. In *MICRO*.

[64] Dam Sunwoo, Gene Y. Wu, Nikhil a. Patil, and Derek Chiou. 2010. PrEsto: An FPGA-accelerated Power Estimation Methodology for Complex Systems. In *FPL*.

[65] Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), 267–288.

[66] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye. 2000. Energy-driven integrated hardware-software optimizations using SimplePower. In *ISCA*.

[67] Thomas F T.F. Wenisch, R.E. Roland E R.E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and James C J.C. Hoe. 2006. SimFlex: Statistical Sampling of Computer System Simulation. *IEEE Micro* 26, 4 (Jul 2006), 18–31.

[68] Gene Wu, Joseph L. Greathouse, Alexander Lyashevsky, Nuwan Jayasena, and Derek Chiou. 2015. GPGPU performance and power estimation using machine learning. In *HPCA*.

[69] Sam Likun Xi, Hans Jacobson, Pradip Bose, Gu-Yeon Wei, and David Brooks. 2015. Quantifying sources of error in McPAT and potential impacts on architectural studies. In *HPCA*.

[70] Xinnian Zheng, Lizy K John, and Andreas Gerstlauer. 2016. Accurate phase-level cross-platform power and performance estimation. In *53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*.

[71] Davide Zoni, Luca Cremona, William Fornaciari, and Milano Dipartimento. 2018. PowerProbe : Run-time Power Modeling Through Automatic RTL Instrumentation. In *DATE*.

[72] Hui Zou and Trevor Hastie. 2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society. Series B: Statistical Methodology* 67, 2 (2005), 301–320.

[73] Hui Zou, Trevor Hastie, and Robert Tibshirani. 2007. On the "degrees of freedom" of the lasso. *Annals of Statistics* 35, 5 (2007), 2173–2192.